Formal Methods for the Certification of Auto-generated Flight Code

Ewen Denney

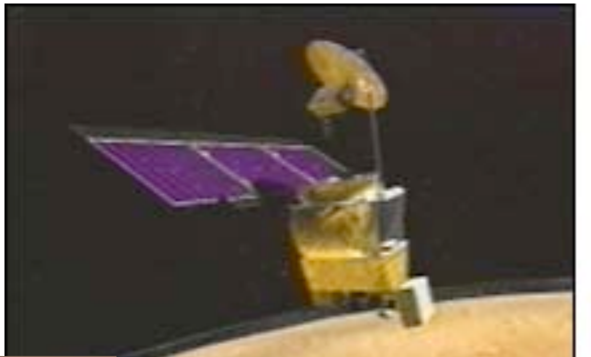
Robust Software Engineering

NASA Ames Research Center

California, USA

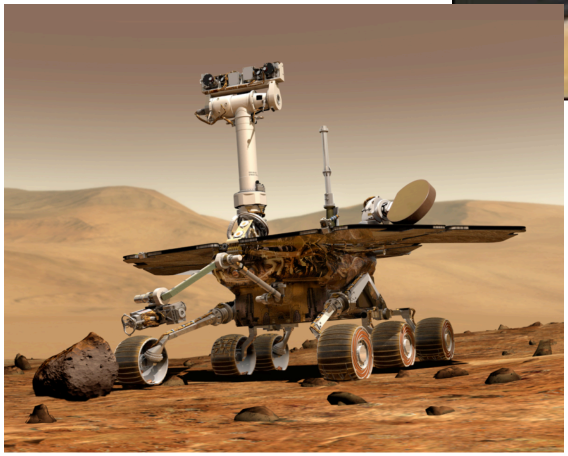
Bugs in Space

- Mars Climate Orbiter (1998)
  - Unit problem in GN&C software
  - Crashed into Mars



(NASA)

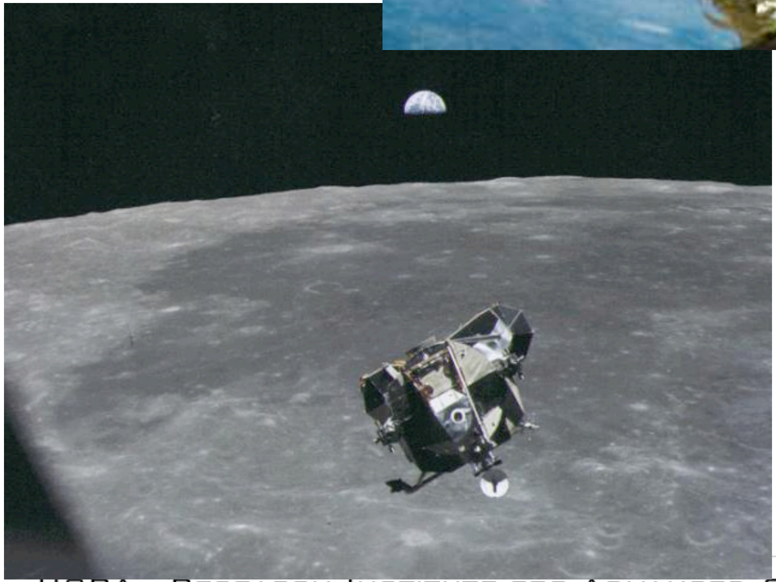
- Mars Polar Lander (1998)
  - Inconsistencies in GN&C landing model
  - Premature engine shut-off
  - Crashed into Mars



- Mars Exploration Rover (2004)
  - Spirit shut down unexpectedly for 10 days
  - Flash ROM overload caused reboot
  - Parameter permitted unlimited consumption of system memory as flash memory was exhausted
  - Lost $4M science a day

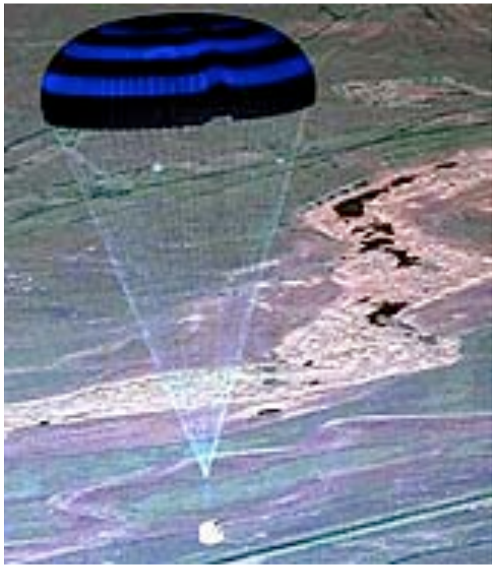
It’s not just now …

- Gemini 5 (1965)
  - Missed landing point by 100 miles
  - GN&C didn’t model rotation of earth around sun



- Apollo 11 (1969)
  - Software reboot during descent to lunar surface
  - Forced manual landing of lunar lander

It’s not just NASA …



Soyuz TMA-1 (2003)

- Missed landing point by 275 miles
- “glitch in the craft’s guidance software”

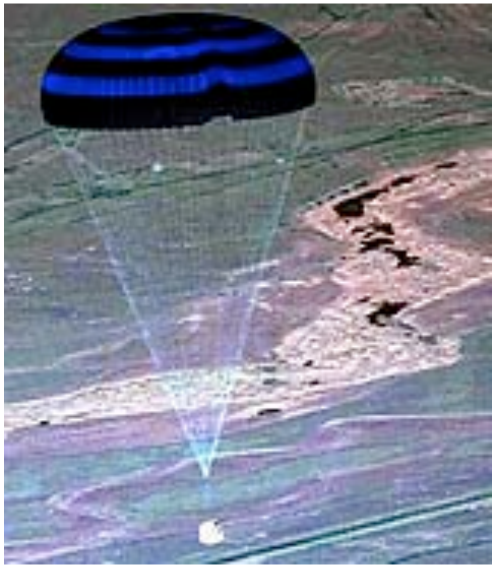
Ariane 5 (1996)

- Bad 64 to 16 bit conversion led to overflow in GN&C software
- Veered off trajectory
- Self-destructed

Cryosat (2005)

- ESA Earth Explorer Mission to measure polar ice
- Launcher fell into ocean when fuel ran out
- Due to “software glitch” in control system: software failed to send command for 2nd stage separation (but “rocket is ok”)

It’s not just NASA …



Soyuz TMA-1 (2003)

- Missed landing point by 275 miles
- “glitch in the craft’s guidance software”

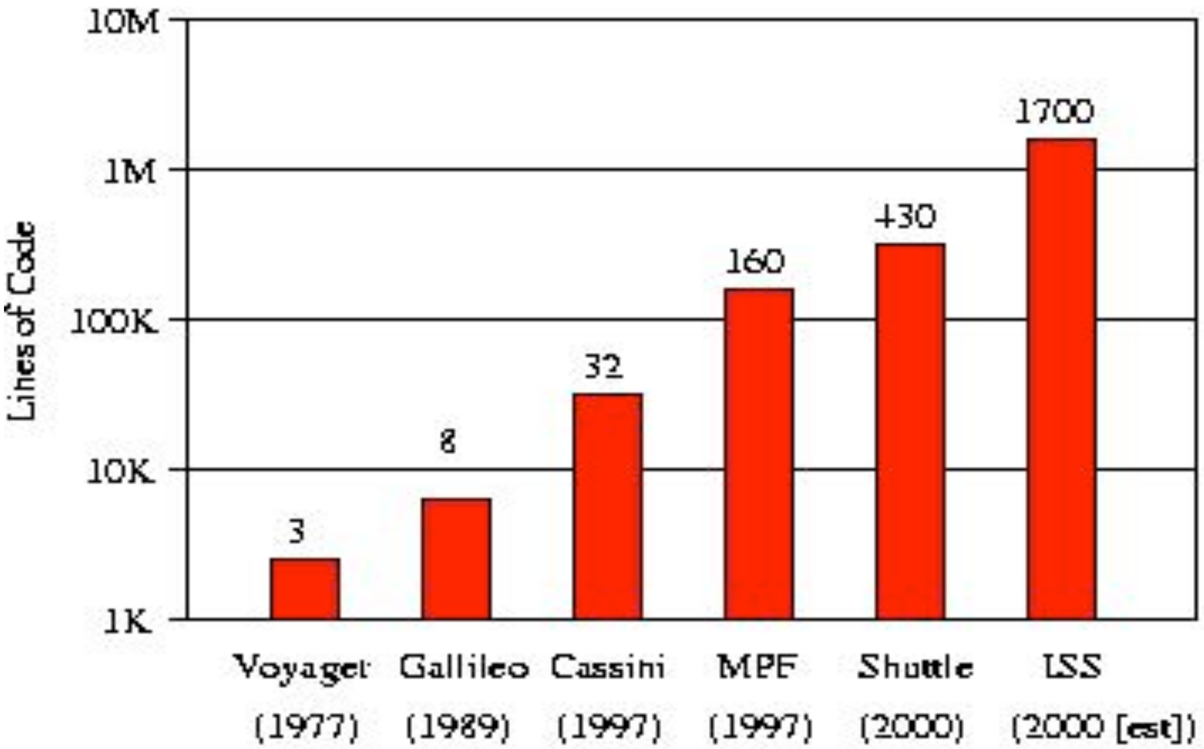
Ariane 5 (1996)

- Bad 64 to 16 bit conversion led to overflow in GN&C software
- Veered off trajectory
- Self-destructed

Cryosat (2005)

- ESA Earth Explorer Mission to measure polar ice
- Launcher fell into ocean when fuel ran out
- Due to “software glitch” in control system: software failed to send command for 2nd stage separation (but “rocket is ok”)

It’s getting worse …



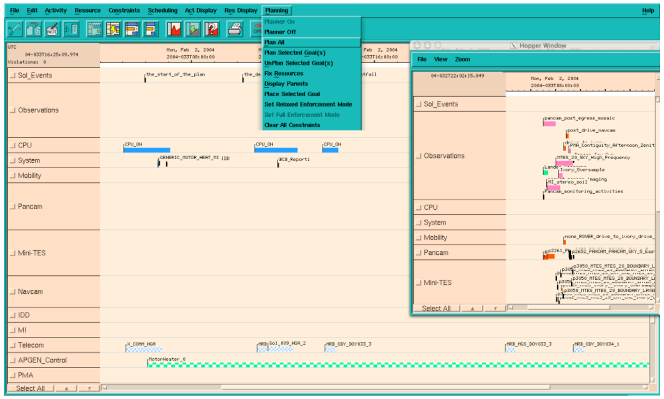
Software for International Space Station (ISS) now estimated at 6.5 MLOC !

… and worse!

Future missions call for vastly increased levels of *intelligence*

Automated planning

- On-board decision-making
  - Spacecraft operations
  - System management
- Schedule generation
  - Crew, equipment, systems



Informed Logistics

- Modeling of failure mechanisms
- Prognostics
- Troubleshooting assistance
- Maintenance planning
- End-of-life decisions

Real-Time Systems Health Management

- Distributed sensing for structural health
- Fault detection, isolation, and recovery
- Failure prediction and mitigation
- Crew and operator interfaces

Adaptive Control

- Improving safety and control performance beyond human ability
- Control in situations of failure or component degradation
- Operability in unknown or changing environments

The million $ question

Given increasingly

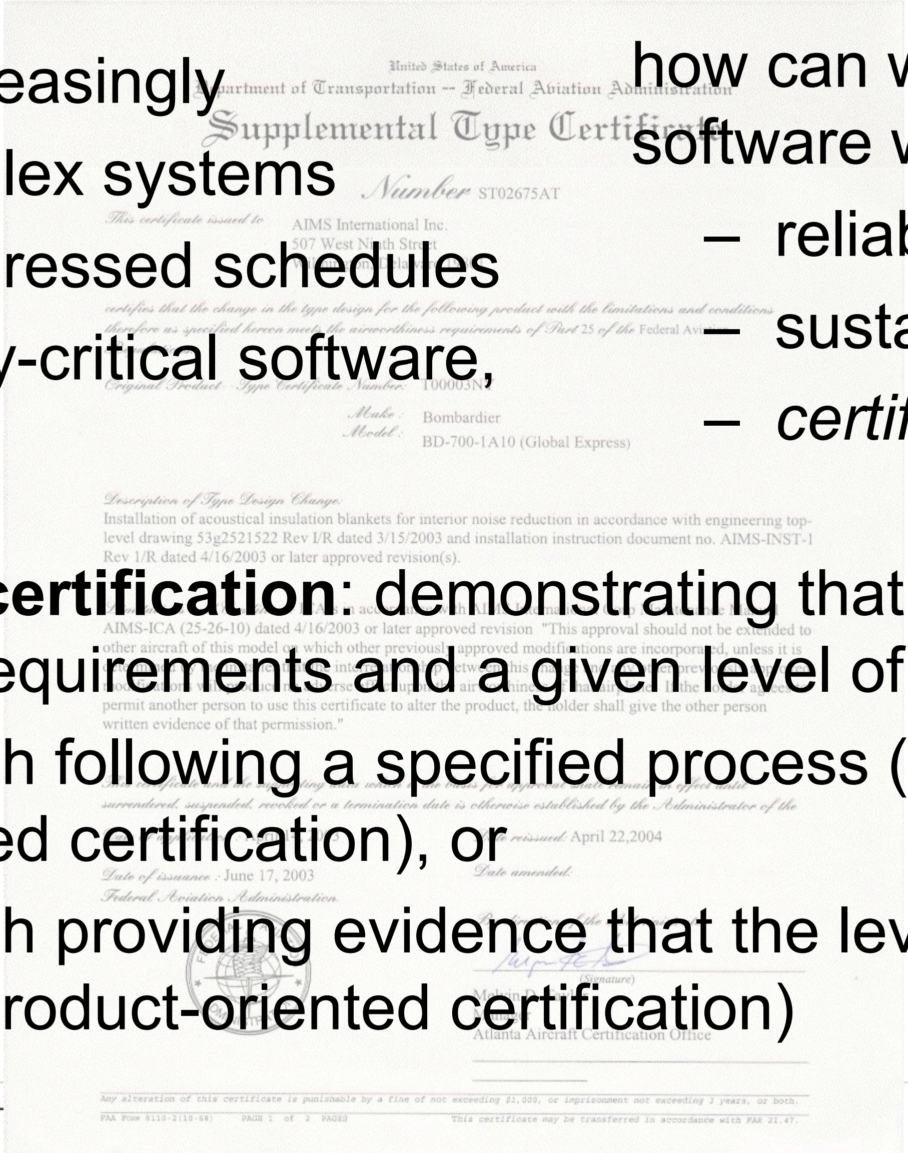
- complex systems
- compressed schedules
- safety-critical software,

how can we develop software which is

- reliable,
- sustainable, and
- *certifiable*?

Software certification: demonstrating that software meets its requirements and a given level of safety, either:

- through following a specified process (process-oriented certification), or
- through providing evidence that the level of safety is met (product-oriented certification)



United States of America
[illegible] Department of Transportation -- Federal Aviation Administration

Supplemental Type Certifi[illegible]

Number ST02675AT

This certificate issued to AIMS International Inc.
507 West Ni[illegible]th Stre[illegible]
Wi[illegible] Delaware [illegible]

certifies that the change in the type design for the following product with the limitations and conditions therefore as specified hereon meets the airworthiness requirements of Part 25 of the Federal Avi[illegible] [illegible]

Original Product - Type Certificate Number: T00003N[illegible]

Make: Bombardier
Model: BD-700-1A10 (Global Express)

Description of Type Design Change:
Installation of acoustical insulation blankets for interior noise reduction in accordance with engineering top-level drawing 53g2521522 Rev I/R dated 3/15/2003 and installation instruction document no. AIMS-INST-1 Rev 1/R dated 4/16/2003 or later approved revision(s).

[illegible] ICAs in acc[illegible] with AI[illegible] [illegible] Manual AIMS-ICA (25-26-10) dated 4/16/2003 or later approved revision "This approval should not be ex[illegible]ended to other aircraft of this model o[illegible] which other previousl[illegible] approved modifi[illegible]tions are incorpora[illegible]ed, unless it is [illegible] inte[illegible] [illegible]etween this [illegible] previously approved modifications will produce [illegible] adverse effect upon [illegible] air[illegible] of that airplane. If the holder agrees to permit another person to use this certificate to alter the product, the holder shall give the other person written evidence of that permission."

This certificate and the supporting data which is the basis for approval shall remain in effect until surrendered, suspended, revoked or a termination date is otherwise established by the Administrator of the

[illegible] April 1[illegible] 2003 *Date reissued:* April 22,2004

Date of issuance: June 17, 2003 *Date amended:*

Federal Aviation Administration

By direction of the Administrator

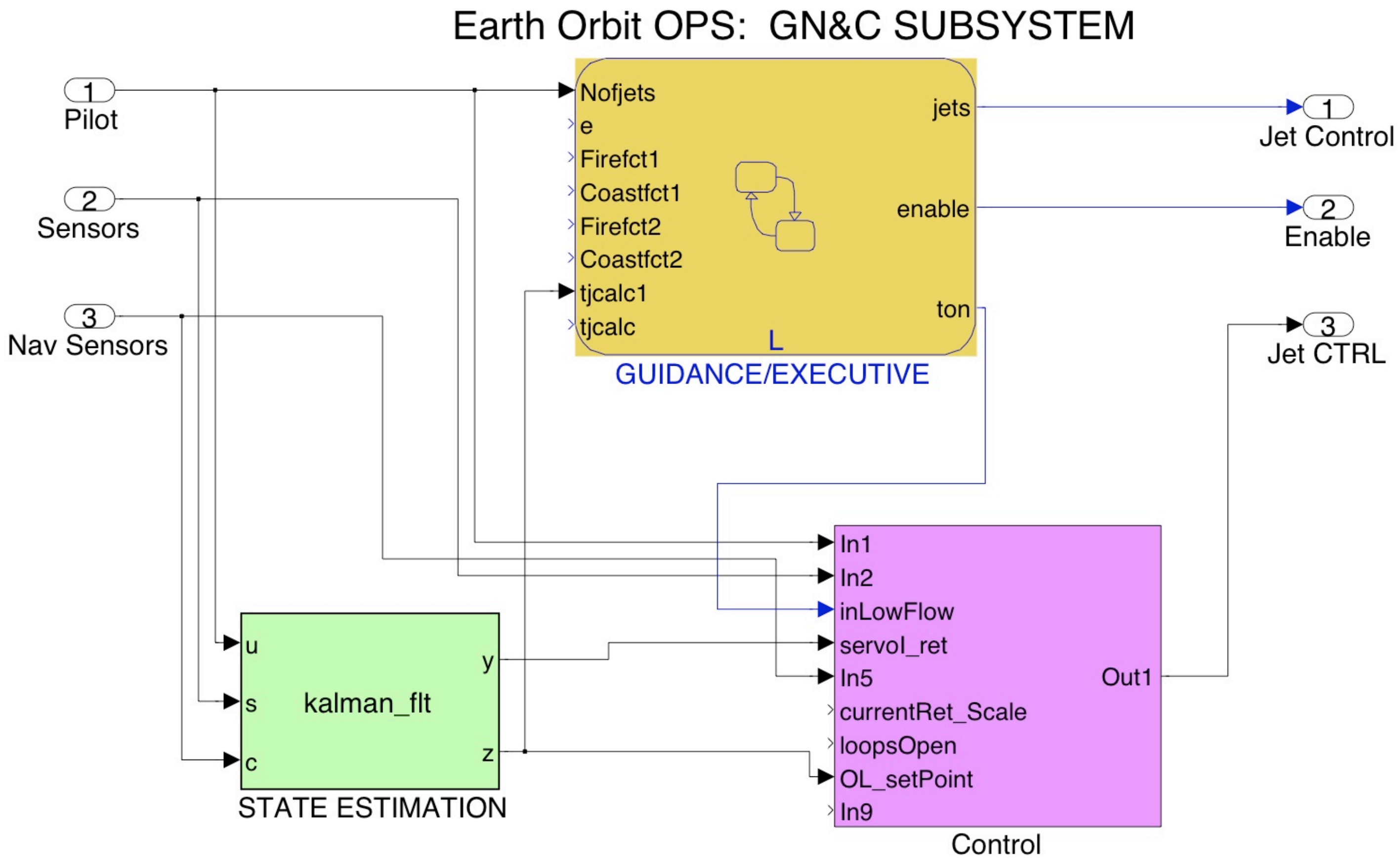
(Signature)

Melvin D. [illegible]
Manager
Atlanta Aircraft Certification Office

Any alteration of this certificate is punishable by a fine of not exceeding $1,000, or imprisonment not exceeding 3 years, or both.

FAA Form 8110-2(10-68) PAGE 1 of 2 PAGES This certificate may be transferred in accordance with FAR 21.47.

Model-based Development



Modeling:

Simulink (control), Stateflow (executive), Embedded Matlab (everywhere)

Code generation: Real-Time Workshop

Code generators are not perfect

Dear …,

If you are using R14SP3 Simulink code generation products, please review the following information. If you are not using R14SP3 versions of MathWorks products, please disregard this message.

We have identified bugs in R14SP3 Simulink^® code generation products, which in rare instances generate incorrect code that is not easily detected. These bugs have been fixed in subsequent releases: R2006a, R2006b, or the upcoming R2007a release.

To prevent impact from these bugs, R14SP3 code generation software users should take the following actions:

Review Related Bug Reports with Potential Workarounds
You can find the documented issues and potential workarounds through the following links (login required):

Bug Report 275411 <http://www.mathworks.com/support/bugreports/details.html?rp=275411>
Bug Report 283331 <http://www.mathworks.com/support/bugreports/details.html?rp=283331>
Bug Report 284002 <http://www.mathworks.com/support/bugreports/details.html?rp=284002>
Bug Report 291423 <http://www.mathworks.com/support/bugreports/details.html?rp=291423>
Bug Report 291978 <http://www.mathworks.com/support/bugreports/details.html?rp=291978>

Frequent updates - bug reports, work-arounds and fixes

Qualification

- A code generator is *qualified*
  - with respect to a given standard
  - for a given project

  if there is sufficient evidence about the generator itself so that V&V need not be carried out on the generated code to certify it
- Must be done for every project, version
  - can obtain *verification credit*
- Generators are rarely qualified
  - ASCET-SE (IEC 61508), SCADE, VAPS (DO-178B)
- Qualifying code generators is (almost) infeasible!

Issues

Commercial code generators are black boxes

  – Not qualified so need to analyze generated code

- Historically buggy: despite extensive heritage, rare bugs still remain
- Cannot detect many bugs at model level or via simulation
- Math intensive code requires powerful analysis techniques

Product-oriented certification

- Augment code generator to generate *certificates* together with code (aka. the "verifying compiler" approach)
- No need to qualify/re-qualify code generator
- Code certificates:
  - proof of a specific safety property
  - can be independently verified
  - require only a small trusted infrastructure
  - process is completely automated
- Support engineers doing software assurance
  - generate safety documentation for human analysts

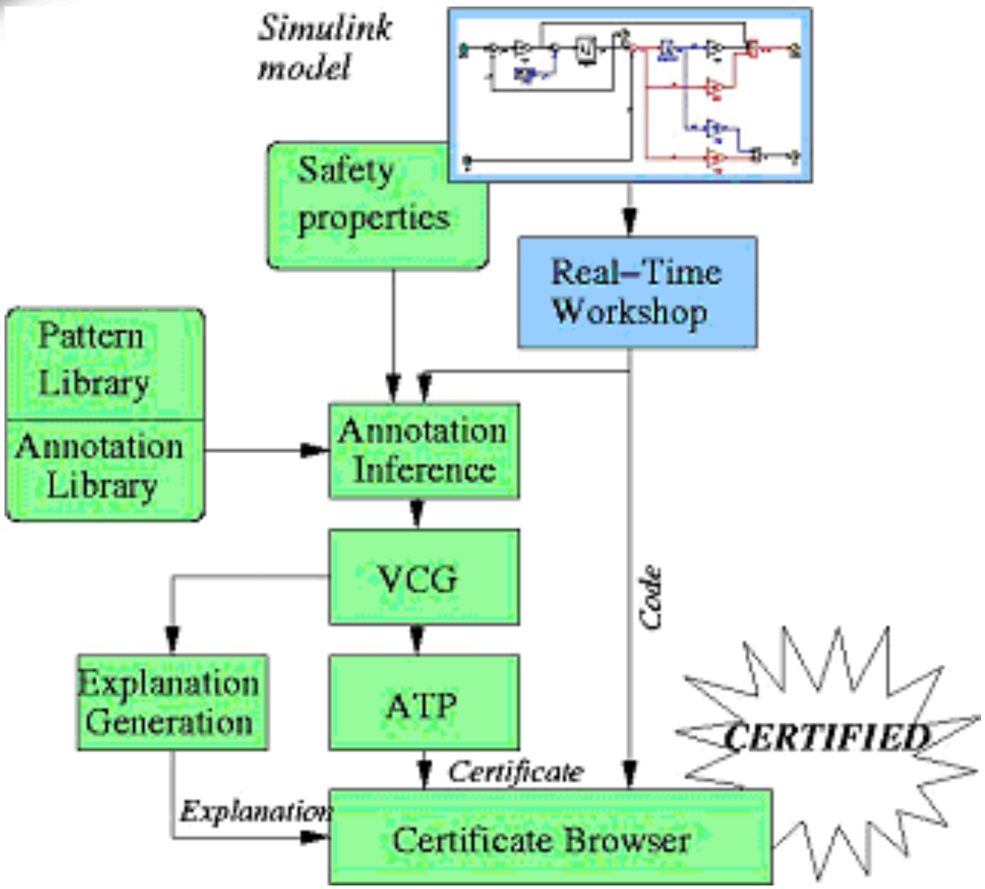
Assurance strategies for autocoding

- Documentation
  - *explain* the code synthesis and certification process
  - increases transparency and trust in process
- Traceability
  - *link* elements of generation process
  - mandated by NASA standards
- Proof
  - mathematical *proof* is gold standard
  - difficult to achieve and interpret without automation
  - show incrementally for individual properties

Use code generator plug-in to automate this

⇒ minimal impact to existing process

Technical approach



- Combine generator with certification plug-in: AutoCert
- Generate certificates which can be verified independently (IV&V)
- Based on formal logic
  - Range of safety properties
  - Pattern-based approach to inferring annotations
  - Fully automated
  - Can be used to generate explanations
  - Small set of trusted components

Language-specific safety properties

Language-specific safety properties to check specific constructs of the target language (C)

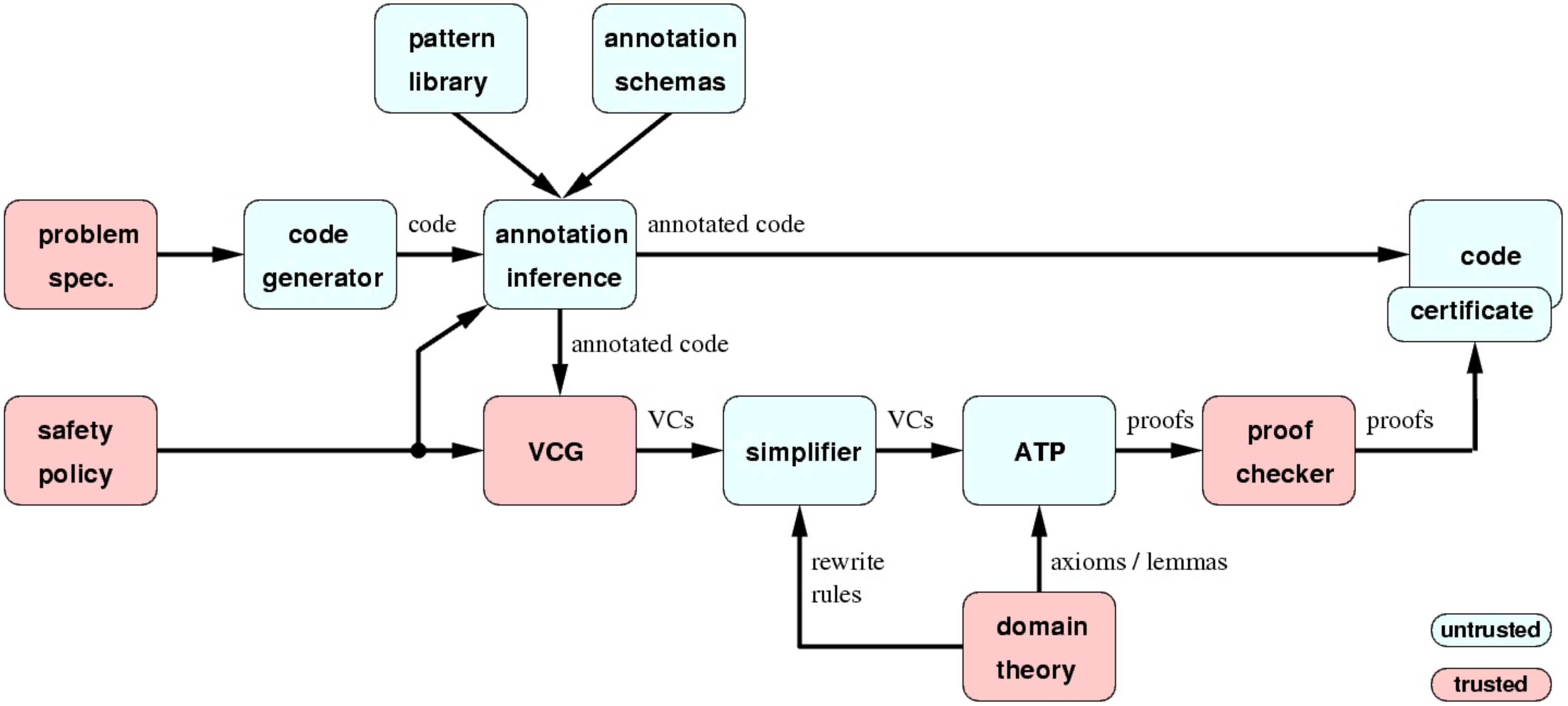
- Memory safety: array bounds
  - Buffer overflows often lead to unsafe programs
- Variable initialization before use
  - Un-initialized variables can cause unpredictable/unrepeatable effects
  - Compilers only check for initialization of scalars
  - e.g.: RTW Bug: uninit variables in DEMUX blocks

Domain-specific safety properties

- invariants
  - matrix symmetry: “covariance matrix P always symmetric”
  - quaternion, probability vector norms: “must add up to 1”
  - coordinate systems, units: “must use consistently”
  - arithmetic saturation/variable ranges: “actuator rate < 0.1”
- system-specific properties
  - “are all sensor data used?”
- block properties
  - “all values of *x* in interpolation table disjoint and increasing”

Simulink control models often “math-heavy”

Trusted Architecture

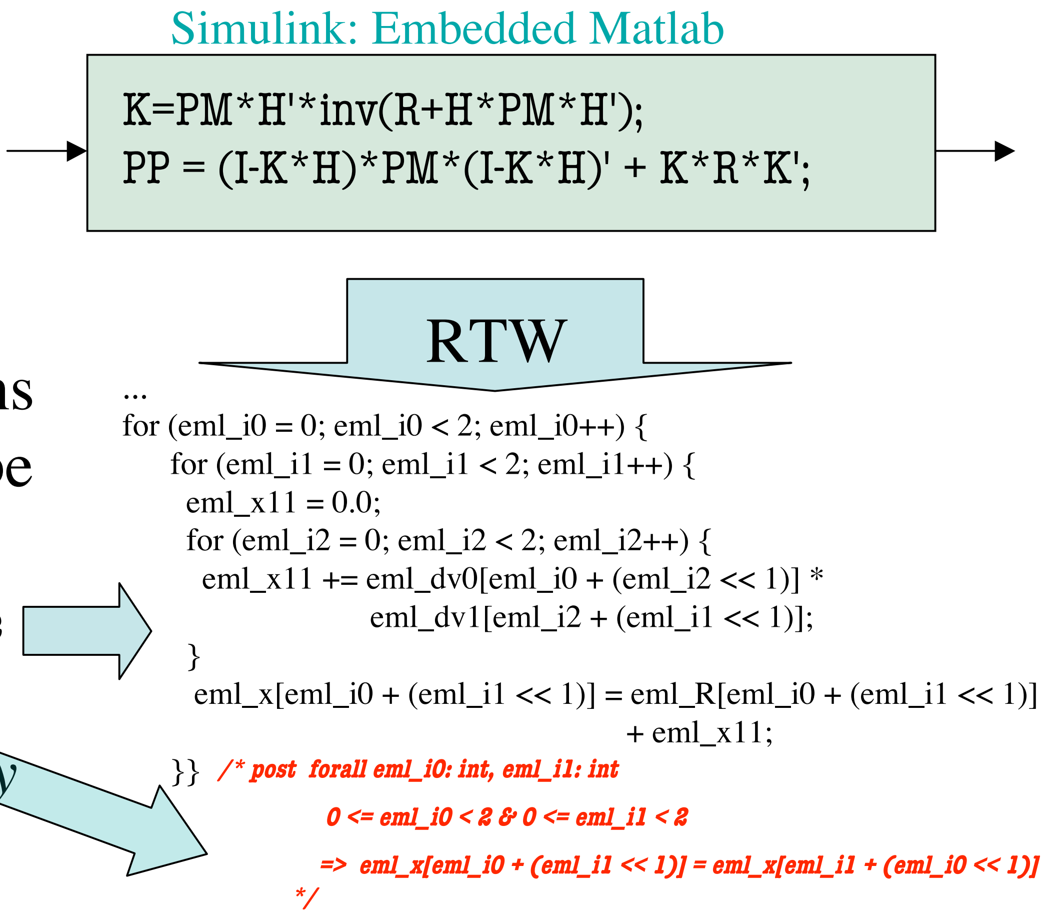


- Small kernel of untrusted components

  - patterns and annotations *untrusted*

Matrix symmetry

- Covariance matrices (PM, PP) in a Kalman filter must remain symmetric during update
- individual matrix operations in the generated code must be checked:
    - `x=R+H*PM*H'` is symmetric
- Annotations are required
- Analysis tool *automatically* generates annotations based upon idiomatic code patterns



Simulink: Embedded Matlab

```
K=PM*H'*inv(R+H*PM*H');
PP = (I-K*H)*PM*(I-K*H)' + K*R*K';
```

RTW

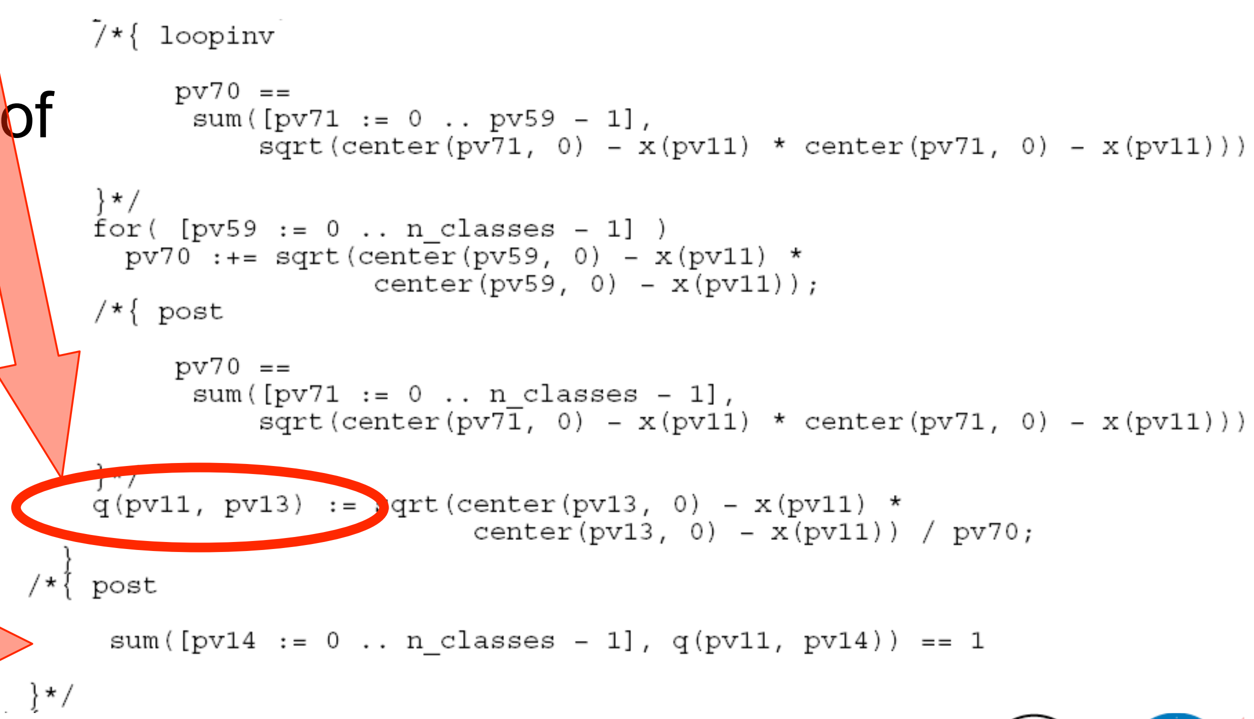
```
...
for (eml_i0 = 0; eml_i0 < 2; eml_i0++) {
     for (eml_i1 = 0; eml_i1 < 2; eml_i1++) {
       eml_x11 = 0.0;
       for (eml_i2 = 0; eml_i2 < 2; eml_i2++) {
         eml_x11 += eml_dv0[eml_i0 + (eml_i2 << 1)] *
                          eml_dv1[eml_i2 + (eml_i1 << 1)];
       }
        eml_x[eml_i0 + (eml_i1 << 1)] = eml_R[eml_i0 + (eml_i1 << 1)]
                                                    + eml_x11;
}}  /* post  forall eml_i0: int, eml_i1: int
                0 <= eml_i0 < 2 & 0 <= eml_i1 < 2
               =>  eml_x[eml_i0 + (eml_i1 << 1)] = eml_x[eml_i1 + (eml_i0 << 1)]
            */
```

Vector norms

- Intuitively:

  Vectors must be normalized

- Show preservation of norm by update operations
- Domain-specific requirement
- Requires code annotations



```
/*{ loopinv

     pv70 ==
      sum([pv71 := 0 .. pv59 - 1],
          sqrt(center(pv71, 0) - x(pv11) * center(pv71, 0) - x(pv11)))

}*/
for( [pv59 := 0 .. n_classes - 1] )
  pv70 :+= sqrt(center(pv59, 0) - x(pv11) *
                center(pv59, 0) - x(pv11));
/*{ post

     pv70 ==
      sum([pv71 := 0 .. n_classes - 1],
          sqrt(center(pv71, 0) - x(pv11) * center(pv71, 0) - x(pv11)))

}*/
q(pv11, pv13) := sqrt(center(pv13, 0) - x(pv11) *
                      center(pv13, 0) - x(pv11)) / pv70;
}
/*{ post

  sum([pv14 := 0 .. n_classes - 1], q(pv11, pv14)) == 1

}*/
```

Traceability

- Traceability:

  “the ability to link requirements back to rationales and forward to corresponding design artifacts, code, and test cases”

Traceability

- Traceability:

  “the ability to link requirements back to rationales and forward to corresponding design artifacts, code, and *proofs*”

- “why is this line of code safe?”

  line of code → verification conditions

- “where does this condition come from?”

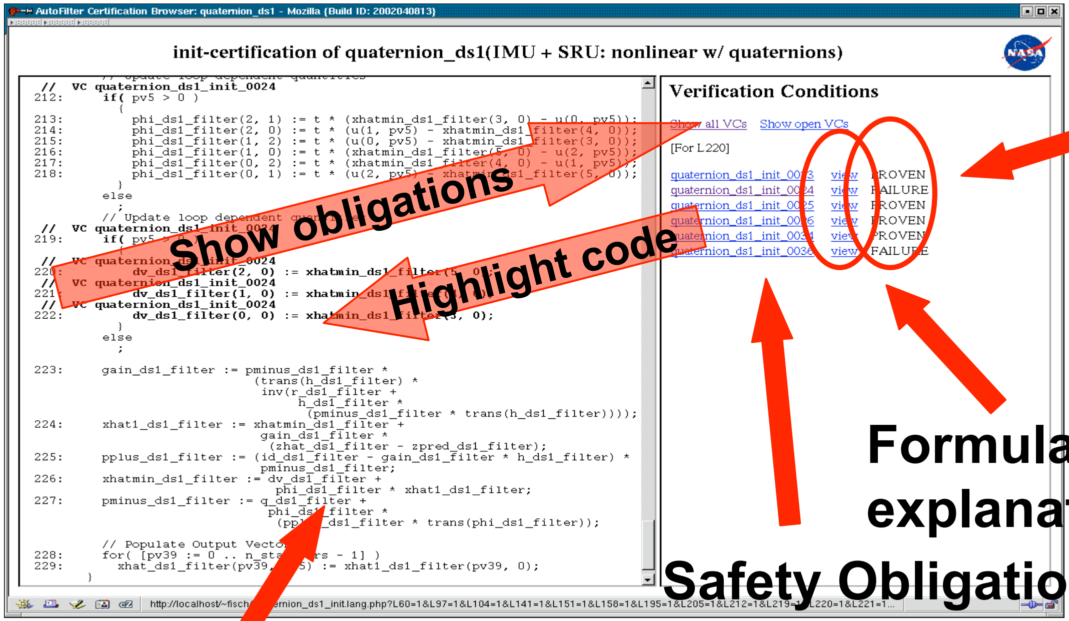
  verification condition → lines of code

Autocode safety reports

- Verification says *that* the code is safe
- Explanation says *why* the code is safe
- Use code analysis to generate *safety report:* explain how code complies with safety properties

  "the variable rtb_GetVeci is in the coordinate frame *Earth-Centric Inertial* because it is defined by applying the ECEF to ECI transformation to the variable … which is in turn…"

  ⇒ support code reviews
- Trace to relevant code fragments and model


AutoFilter Certification Browser: quaternion_ds1 - Mozilla {Build ID: 2002040813}
Show obligations
Highlight code

init-certification of quaternion_ds1(IMU + SRU: nonlinear w/ quaternions)

```
        // Update loop dependent quantities
 //  VC quaternion_ds1_init_0024
212:      if( pv5 > 0 )
            {
213:          phi_ds1_filter(2, 1) := t * (xhatmin_ds1_filter(3, 0) - u(0, pv5));
214:          phi_ds1_filter(2, 0) := t * (u(1, pv5) - xhatmin_ds1_filter(4, 0));
215:          phi_ds1_filter(1, 2) := t * (u(0, pv5) - xhatmin_ds1_filter(3, 0));
216:          phi_ds1_filter(1, 0) := t * (xhatmin_ds1_filter(5, 0) - u(2, pv5));
217:          phi_ds1_filter(0, 2) := t * (xhatmin_ds1_filter(4, 0) - u(1, pv5));
218:          phi_ds1_filter(0, 1) := t * (u(2, pv5) - xhatmin_ds1_filter(5, 0));
            }
          else
            ;
          // Update loop dependent quantities
 //  VC quaternion_ds1_init_0024
219:      if( pv5 > 0 )
            {
 //  VC quaternion_ds1_init_0024
220:          dv_ds1_filter(2, 0) := xhatmin_ds1_filter(5, 0);
 //  VC quaternion_ds1_init_0024
221:          dv_ds1_filter(1, 0) := xhatmin_ds1_filter(4, 0);
 //  VC quaternion_ds1_init_0024
222:          dv_ds1_filter(0, 0) := xhatmin_ds1_filter(3, 0);
            }
          else
            ;

223:      gain_ds1_filter := pminus_ds1_filter *
                             (trans(h_ds1_filter) *
                              inv(r_ds1_filter +
                                  h_ds1_filter *
                                   (pminus_ds1_filter * trans(h_ds1_filter))));
224:      xhat1_ds1_filter := xhatmin_ds1_filter +
                              gain_ds1_filter *
                               (zhat_ds1_filter - zpred_ds1_filter);
225:      pplus_ds1_filter := (id_ds1_filter - gain_ds1_filter * h_ds1_filter) *
                              pminus_ds1_filter;
226:      xhatmin_ds1_filter := dv_ds1_filter +
                                phi_ds1_filter * xhat1_ds1_filter;
227:      pminus_ds1_filter := q_ds1_filter +
                               phi_ds1_filter *
                                (pplus_ds1_filter * trans(phi_ds1_filter));

          // Populate Output Vector
228:      for( [pv39 := 0 .. n_statevars - 1] )
229:        xhat_ds1_filter(pv39, 5) := xhat1_ds1_filter(pv39, 0);
       }
```

Verification Conditions

Show all VCs   Show open VCs

[For L220]

quaternion_ds1_init_0023	view	PROVEN
quaternion_ds1_init_0024	view	FAILURE
quaternion_ds1_init_0025	view	PROVEN
quaternion_ds1_init_0026	view	PROVEN
quaternion_ds1_init_0034	view	PROVEN
quaternion_ds1_init_0036	view	FAILURE

http://localhost/~fisch/quaternion_ds1_init.lang.php?L60=1&L97=1&L104=1&L141=1&L151=1&L158=1&L195=1&L205=1&L212=1&L219=1&L220=1&L221=1...

Proof Status

Formula or explanation

Safety Obligations

Auto-generated code

Summary

- Formal basis
  - Safety requirements in first-order logic
  - Semantics in VCG
  - Prove VCs with ATP
- Tool
  - Tight integration with development tool suite
  - Trace code and model to verification artifacts
  - Trusted architecture
- Usage
  - Incremental approach
  - Generates safety documentation
  - Supports independent V&V